Computer Organization
Introduction

Here are some natural questions:

How does a computer execute a program?

What is there inside a computer?

Do all computer have identical hardware?

What is the difference between a PC and a Mac?
Technologies

A computer is an instruction-execution engine.
Different hardware technologies are possible:

- Mechanical
- Pneumatic
- Electronic
- Quantum
- Biological

We will focus on electronic technology only, which is the most common technology used today. It primarily uses silicon-based integrated circuits.
Classification

General purpose
Your PC

Special purpose
The computers in your car
The computer in your cell phone
The computer inside your camera
The computer in your washing machine
Partial History of modern day computers

Eckert and Mauchley

Moore School of the U. of Pennsylvania, ENIAC

John Von Neumann

Princeton U.

EDVAC, the blueprint of the first stored program digital computer

Maurice Wilkes

Cambridge U., EDSAC, the first operational stored-program digital computer

John Vincent Atanasoff

Iowa State University

Designed a machine in 1939-1940 to solve differential equations. Recognition came much later.
Generations

First generation: vacuum tubes
Second generation: transistors
Third generation: integrated circuits
Fourth generation: LSI and VLSI

Units of time

1 second

1 millisecond (ms) = 10^{-3} second
1 microsecond (µs) = 10^{-6} second
1 nanosecond (ns) = 10^{-9} second
1 picosecond (ps) = 10^{-12} second
Questions

My PC has a 3 GHz clock. What does the clock do?

How much time does it take to add two integers?

How much time does your computer take to read a 1 MB (megabyte) file from a disk?

What distance does an electronic signal travel in 1 nanosecond?
A Basic Digital Computer

There are different ways of designing the “boxes” or the functional units. At the upper level, we care only about the functionality and not so much about their internal construction.
**Measuring the Speed**

MIPS = Million Instructions Per Second

MFLOPS = Million FLOating point ops Per Sec

GFLOPS = Billion (Giga) FLOating point ops Per Sec

TERAFLOPS = Trillion FLOating point ops Per Sec

PETAFLOPS = $10^{15}$ FLOating point ops Per Sec

What do we do with a TERAFLOP or a PETAFLOP machine? Do we have enough work for them (other than playing video games)?
Laws of Hardware

• Signals cannot travel faster than the speed of light.
• Memory is always slower than the CPU.
• Software is slower than hardware.

Moore’s Law.
The packaging density of transistors on an integrated circuit increases $2x$ every 18 months.

Gates Law.
The speed of software halves every 18 months
(Microsoft is the worst offender. Software bloat almost compensates for hardware improvement due to Moore’s law).

Amdahl’s law
Concerned with the speedup achievable from an improvement to a computation that affects a fraction of that computation.
Factors influencing computer performance

How fast can you solve a problem on a machine?

Depends on

- The algorithm used
- The HLL program code
- The efficiency of the compiler

And, of course, it also depends on the target machine. If the algorithm is lousy, then do not blame the computer!
Assembly Language Programming
Program a robot

It should move on a 2D plane, or sometimes jump (without moving)

What should a typical program look like?
How will you encode it?
How will the robot understand your language?
High-level vs. Assembly language

Consider the following statements

1. \( a = x + y - z \)
2. if \( x > y \)
   then \( x := x + y \)
   else \( x := x - y \)

HLL (High Level Language) programs are machine independent. They are easy to learn, easy to use, and convenient for managing complex tasks.

Assembly language programs are machine specific. It is the language that the processor “directly” understands.
Memory can be viewed as a bookshelf.

View registers as spaces on your table.
Understanding Assembly Language

Let us begin with data representation. How to represent

- Signed integers
- Fractions
- Alphanumeric characters
- Floating point numbers
- Pictures?

Memory

| 0 1 0 0 1 0 1 1 |
| 1 1 0 1 1 0 1 0 |
| 1 0 0 1 1 0 0 0 |

Can you read the contents of these memory cells?
Visualizing instruction execution

(The main concept is register-transfer operation.

A register is a fast storage within the CPU

\[ a = x + y - z \]

\[
\begin{align*}
\text{load } x \text{ into } r1 \\
\text{load } y \text{ into } r2 \\
\text{load } z \text{ into } r0 \\
\text{r3 } & \leftarrow r1 + r2 \\
\text{r0 } & \leftarrow r3 - r0 \\
\text{store } r0 \text{ into } a
\end{align*}
\]
Assembly language instructions for a hypothetical machine (not MIPS)

Load x, r1
Load y, r2
Load z, r0
Add r3, r1, r2
Sub r0, r3, r0
Store r0, a

Each processor has a different set of registers, and different assembly language instructions. The assembly language instructions of Intel Pentium and MIPS are completely different.

Motorola 68000 has 16 registers r0-r15
MIPS has 32 registers r0-r31
Pentium has 8 general purpose & 6 segment registers.
**Binary or Machine Language program**

Both program and data are represented using **only 0’s and 1’s** inside a computer. Here is a sample:

These are **instruction formats**. Each instruction has a specific format.
Can we distinguish program from data?

Both are bit strings.
Indistinguishable.

MEMORY

Normally, the programmer has to tell the machine (or use some convention) to specify the address of the first instruction. Incorrect specification will lead to errors, and the program is most likely to crash.
Bits, bytes, words

Bit: 0, 1

Byte: string of 8 bits. Each byte has an address.

Word: one or more bytes (usually 2 or 4 or 8).
### Byte order in a word

<table>
<thead>
<tr>
<th>0</th>
<th>Word 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Word 1</td>
</tr>
<tr>
<td>8</td>
<td>Word 2</td>
</tr>
<tr>
<td>12</td>
<td>Word 3</td>
</tr>
</tbody>
</table>

- **Big Endian order**  
  - [byte 0, byte 1, byte 2, byte 3]

- **Little Endian order**  
  - [byte 3, byte 2, byte 1, byte 0]
Registers vs. memory

Data can be stored in registers or memory locations. Memory access is slower (takes approximately 50 ns) than register access (takes approximately 1 ns or less).

To increase the speed of computation it pays to keep the variables in registers as long as possible. However, due to technology limitations, the number of registers is quite limited (typically 8-64).

MIPS registers

MIPS has 32 registers r0-r31. The conventional use of these registers is as follows:
<table>
<thead>
<tr>
<th>register</th>
<th>assembly name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>$zero</td>
<td>Always 0</td>
</tr>
<tr>
<td>r1</td>
<td>$at</td>
<td>Reserved for assembler</td>
</tr>
<tr>
<td>r2-r3</td>
<td>$v0-$v1</td>
<td>Stores results</td>
</tr>
<tr>
<td>r4-r7</td>
<td>$a0-$a3</td>
<td>Stores arguments</td>
</tr>
<tr>
<td>r8-r15</td>
<td>$t0-$t7</td>
<td>Temporaries, not saved</td>
</tr>
<tr>
<td>r16-r23</td>
<td>$s0-$s7</td>
<td>Contents saved for later use</td>
</tr>
<tr>
<td>r24-r25</td>
<td>$t8-$t9</td>
<td>More temporaries, not saved</td>
</tr>
<tr>
<td>r26-r27</td>
<td>$k0-$k1</td>
<td>Reserved by operating system</td>
</tr>
<tr>
<td>r28</td>
<td>$gp</td>
<td>Global pointer</td>
</tr>
<tr>
<td>r29</td>
<td>$sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>r30</td>
<td>$fp</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>r31</td>
<td>$ra</td>
<td>Return address</td>
</tr>
</tbody>
</table>
Example assembly language programs

Example 1

\[
\text{f} = \text{g} + \text{h} - \text{i}
\]

Assume that f, g, h, i are assigned to $s0, $s1, $s2, $s3

\[
\begin{align*}
\text{add} & \quad \text{$t0, $s1, $s2} & \quad \# \text{register $t0$ contains g + h} \\
\text{sub} & \quad \text{$s0, $t0, $s3} & \quad \# \text{f = g + h - i}
\end{align*}
\]

Example 2

\[
\text{g} = \text{h} + \text{A[8]}
\]

Assume that g, h are in $s1, $s2. A is an array of words, the elements are stored in consecutive locations of the memory. The base address is stored in $s3.

\[
\begin{align*}
\text{l}w & \quad \text{t0, 32($s3)} & \quad \# \text{t0 gets A[8], 32= 4x 8} \\
\text{add} & \quad \text{$s1, $s2, $t0} & \quad \# \text{g = h + A[8]}
\end{align*}
\]
**Machine language representations**

Instruction “add” belongs to the R-type format.

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shift amt</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

The function field is an extension of the opcode, and they together determine the operation.

Note that “sub” has a similar format.